Algorithmic Game Theory

Paul W. Goldberg[1]

[1]Department of Computer Science
Oxford University, U. K.

STACS'15 tutorial, Munich
March 2015

Topics

Mainly, complexity of equilibrium computation...

- Problem statements, Nash equilibrium
- **NP**-completeness of finding certain Nash equilibria[1]
- Total search problems, **PPAD** and related complexity classes
- **PPAD**-completeness of finding unrestricted Nash equilibria [2]
- Computation of *approximate* Nash equilibria
- models for "constrained" computation of NE/CE: communication-bounded, query-bounded

Apology: I won't cover potential games/**PLS**, and various other things

[1] I will give you definitions soon!

[2] Daskalakis, G, Papadimitriou: The Complexity of Computing a Nash equilibrium. SICOMP/CACM Feb'09.
Chen, Deng, Teng: Settling the complexity of computing two-player Nash equilibria. JACM, 2009.

Game Theory and Computer Science

- Modern CS and GT originated with John von Neumann at Princeton in the 1950's (Yoav Shoham: Computer Science and Game Theory. *CACM* Aug'08.))
- Common motivations:
  - modeling *rationality* (interaction of selfish agents on Internet);
  - AI: solve cognitive tasks such as negotiation

Game Theory and Computer Science

- Modern CS and GT originated with John von Neumann at Princeton in the 1950's (Yoav Shoham: Computer Science and Game Theory. *CACM* Aug'08.))
- Common motivations:
  - modeling *rationality* (interaction of selfish agents on Internet);
  - AI: solve cognitive tasks such as negotiation

- It turns out that GT gives rise to problems that pose very interesting mathematical challenges, e.g. w.r.t. computational complexity. Complexity classes **PPAD** and **PLS**.

	cooperate	defect
cooperate	8 8	10 0
defect	0 10	1 1

There's a *row player* and a *column player*.

Nash equilibrium: no incentive to change

Example 1: Prisoners' dilemma

		cooperate 0	defect 1
cooperate	0	8 / 8	10 / 0
defect	1	0 / 10	1 / 1

There's a *row player* and a *column player*.

Solution: both players defect. Numbers in red are probabilities.

Nash equilibrium: no incentive to change

Example 2: Rock-paper-scissors

2008 Rock-paper-scissors Championship (Las Vegas, USA)

Rock-paper-scissors: payoff matrix

	rock	paper	scissors
rock	0, 0	-1, 1	1, -1
paper	1, -1	0, 0	-1, 1
scissors	-1, 1	1, -1	0, 0

Rock-paper-scissors: payoff matrix

		rock 1/3	paper 1/3	scissors 1/3
rock	1/3	0, 0	-1, 1	1, -1
paper	1/3	1, -1	0, 0	-1, 1
scissors	1/3	-1, 1	1, -1	0, 0

Solution: both players randomize: probabilities are shown in red.

Rock-paper-scissors: a non-symmetrical variant

	rock	paper	scissors
rock	0 0	1 -1	-1 **2**
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

What is the solution?

Rock-paper-scissors: a non-symmetrical variant

		rock 1/3	paper 5/12	scissors 1/4
rock	1/3	0 0	1 -1	-1 **2**
paper	1/3	-1 1	0 0	1 -1
scissors	1/3	1 -1	-1 1	0 0

What is the solution?
(thanks to Rahul Savani's on-line Nash equilibrium solver.)

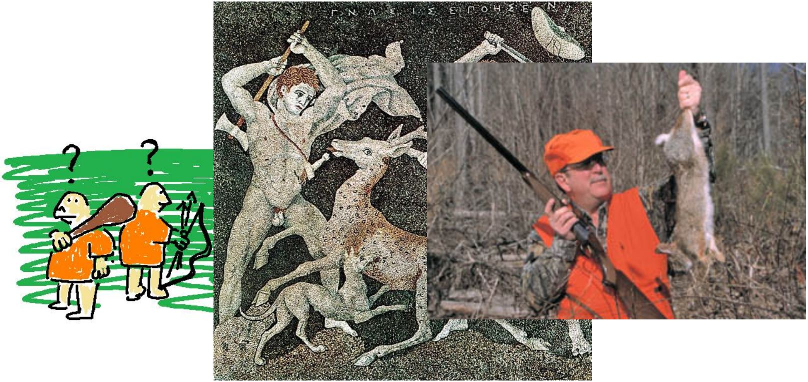
Example 3: Stag hunt

2 hunters; each chooses whether to hunt stag or rabbit...

Example 3: Stag hunt

2 hunters; each chooses whether to hunt stag or rabbit...
It takes 2 hunters to catch a stag,

Example 3: Stag hunt



2 hunters; each chooses whether to hunt stag or rabbit...
It takes 2 hunters to catch a stag, but only one to catch a rabbit.

Stag hunt: payoff matrix

		hunt stag 1	hunt rabbit 0
hunt stag	1	8, 8	0, 1
hunt rabbit	0	1, 0	1, 1

Solution: both hunt stag (the *best* solution).

Stag hunt: payoff matrix

		hunt stag	hunt rabbit
		0	1
hunt stag	0	8, 8	0, 1
hunt rabbit	1	1, 0	1, 1

Solution: both hunt stag (the *best* solution). Or, both players hunt rabbit.

Stag hunt: payoff matrix

		hunt stag 1/8	hunt rabbit 7/8
hunt stag	1/8	8 8	1 0
hunt rabbit	7/8	0 1	1 1

Solution: both hunt stag (the *best* solution). Or, both players hunt rabbit. Or, both players randomize (with the right probabilities).

Nash equilibrium; general motivation

- it should specify a strategy for each player, such that each player is receiving optimal payoff in the context of the other players' choices.

John Forbes Nash

Nash equilibrium; general motivation

- it should specify a strategy for each player, such that each player is receiving optimal payoff in the context of the other players' choices.
- A pure Nash equilibrium is one in which each player chooses a pure strategy — problem: for some games, there is no pure Nash equilibrium!

John Forbes Nash

Nash equilibrium; general motivation

- it should specify a strategy for each player, such that each player is receiving optimal payoff in the context of the other players' choices.
- A pure Nash equilibrium is one in which each player chooses a pure strategy — problem: for some games, there is no pure Nash equilibrium!
- A mixed Nash equilibrium assigns, for each player, a *probability distribution* over his pure strategies, so that a player's payoff is his *expected* payoff w.r.t. these distributions — Nash's theorem shows that this always exists! **Every game has an outcome— as required** Generally, an odd number of equilibria. I return to this later, it is important

John Forbes Nash

Definition and notation

Game: set of players, each player has his own set of allowed actions (also known as "pure strategies"). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Definition and notation

Game: set of players, each player has his own set of allowed actions (also known as "pure strategies"). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)
Number the players $1, 2, \ldots, k$.

Definition and notation

Game: set of players, each player has his own set of allowed actions (also known as "pure strategies"). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Number the players $1, 2, ..., k$.

Let S_p denote player p's set of actions. e.g. in rock-paper-scissors, $S_1 = S_2 = \{\text{rock}, \text{paper}, \text{scissors}\}$.

Definition and notation

Game: set of players, each player has his own set of allowed actions (also known as "pure strategies"). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Number the players $1, 2, ..., k$.

Let S_p denote player p's set of actions. e.g. in rock-paper-scissors, $S_1 = S_2 = \{\mathrm{rock}, \mathrm{paper}, \mathrm{scissors}\}$.

n denotes the size of the largest S_p. (So, in rock-paper-scissors, $k = 2$, $n = 3$.) If k is a constant, we seek algorithms polynomial in n. Indeed, much work studies special case $k = 2$, where a game's payoffs can be written down in 2 matrices.

$S = S_1 \times S_2 \times \ldots \times S_k$ is the set of *pure strategy profiles*. i.e. if $s \in S$, s denotes a choice of action, for each player.

Definition and notation

Game: set of players, each player has his own set of allowed actions (also known as "pure strategies"). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Number the players $1, 2, ..., k$.

Let S_p denote player p's set of actions. e.g. in rock-paper-scissors, $S_1 = S_2 = \{\text{rock}, \text{paper}, \text{scissors}\}$.

n denotes the size of the largest S_p. (So, in rock-paper-scissors, $k = 2$, $n = 3$.) If k is a constant, we seek algorithms polynomial in n. Indeed, much work studies special case $k = 2$, where a game's payoffs can be written down in 2 matrices.

$S = S_1 \times S_2 \times \ldots \times S_k$ is the set of *pure strategy profiles*. i.e. if $s \in S$, s denotes a choice of action, for each player.

Each $s \in S$ gives rise to *utility* or *payoff* to each player. u_s^p will denote the payoff to player p when all players choose s.

Definition and notation

Two parameters, k and n.

normal-form game: list of all u_s^p's

- 2-player: 2 $n \times n$ matrices; so $2n^2$ numbers
- k-player: kn^k numbers

...poly for constant k

General issue:

Input: Game; **Output:** NE.

run-time of algorithms in terms of n

k is small constant; often $k = 2$.

When can it be polynomial in n?

Definition and notation

Two parameters, k and n.
normal-form game: list of all u_s^p's

- 2-player: 2 $n \times n$ matrices; so $2n^2$ numbers
- k-player: kn^k numbers

...poly for constant k

General issue:

Input: Game; **Output:** NE.
run-time of algorithms in terms of n
k is small constant; often $k = 2$.
When can it be polynomial in n?

So you want large k? Fixes:

- "concisely represented" multi-player games
- Consider game with "query access" to payoff function

limitations

- The basic model has limited expressive power. In a *Bayesian game*, u_s^p could be probability distribution over p's payoff, allowing one to represent uncertainty about a payoff.
- This is not really intended to describe combinatorial games like chess, where players take turns. One could define a strategy in advance, but it would be impossibly large to represent...
- We are just considering "one shot" games

Computational problem

Pure Nash

Input: A game in normal form, essentially consisting of all the values u_s^p for each player p and strategy profile s.

Question: Is there a **pure Nash equilibrium.**

Computational problem

Pure Nash

Input: A game in normal form, essentially consisting of all the values u_s^p for each player p and strategy profile s.

Question: Is there a **pure Nash equilibrium.**

That decision problem has corresponding search problem that replaces the question with

Output: A pure Nash equilibrium.

If the number of players k is a constant, the above problems are in **P**. If k is not a constant, you should really study "concise representations" of games.

Another computational problem

NASH

Input: A game in normal form, essentially consisting of all the values u^p_s for each player p and strategy profile s.

Output: A (mixed) Nash equilibrium.

By Nash's theorem, intrinsically a search problem, not a decision problem.

Another computational problem

Nash

Input: A game in normal form, essentially consisting of all the values u_s^p for each player p and strategy profile s.
Output: A (mixed) Nash equilibrium.

By Nash's theorem, intrinsically a search problem, not a decision problem.
3+ players: big problem: solution may involve irrational numbers.
Quick/dirty fix: switch to *approximation*:
Replace "no incentive to change" by "low incentive"

Useful Analogy

(total) search for root of (odd-degree) polynomial: look for approximation

Re-state the problem

ϵ**-Nash equilibrium:** Expected payoff $+\epsilon \geq$ exp'd payoff of best possible response

Approximate Nash

Input: A game in normal form, essentially consisting of all the values u_s^p for each player p and strategy profile s.
$u_s^p \in [0, 1]$.
small $\epsilon > 0$

Output: A (mixed) ϵ-Nash equilibrium.

Notice that we restrict payoffs to $[0, 1]$ (why?)
Formulate computational problem as: Algorithm to be polynomial in n and $1/\epsilon$.
If the above is <u>hard</u>, then it's hard to find a true Nash equilibrium.

Computational complexity

Let's think about the distinction between search problems and decision problems.

We still have decision problems like: *Does there exist a mixed Nash equilibrium with total payoff* $\geq \frac{2}{3}$*?*

Polynomial-time reductions

$\mathcal{I}(X)$ denotes instances of problem X
For decision problems, where $x \in \mathcal{I}(X)$ has $output(x) \in \{yes, no\}$,
to reduce X to X',
poly-time computable function $f{:}\mathcal{I}(X) \longrightarrow \mathcal{I}(X')$

$$output(f(x)) = output(x)$$

[3] I should really talk about poly-time checkable relations

Polynomial-time reductions

$\mathcal{I}(X)$ denotes instances of problem X
For decision problems, where $x \in \mathcal{I}(X)$ has $output(x) \in \{yes, no\}$,
to reduce X to X',
poly-time computable function $f : \mathcal{I}(X) \longrightarrow \mathcal{I}(X')$

$$output(f(x)) = output(x)$$

Search problems:
Given $x \in \mathcal{I}(X)$, $output(x)$ is a poly-length string.[3]
Poly-time computable functions

$$f : \mathcal{I}(X) \longrightarrow \mathcal{I}(X') \quad \text{and} \quad g : solutions(X') \longrightarrow solutions(X)$$

If $y = f(x)$ then $g(output(y)) = output(x)$.
This achieves aim of showing that if $X' \in \mathbf{P}$ then $X \in \mathbf{P}$;
equivalently if $X \notin \mathbf{P}$ then $X' \notin \mathbf{P}$.

[3] I should really talk about poly-time checkable relations

All **NP** decision problems have corresponding **NP** search problems where y is certificate of "$output(x) = yes$"
e.g. given boolean formula Φ, is it satisfiable? y is satisfying assignment (which is hard to find but easy to check)
Total search problems (e.g. NASH and others) are more tractable in the sense that for all problem instances x, $output(x) = yes$.
So, every instance has a solution, and a certificate.

NP-Completeness of finding "good" Nash equilibria

2-player game: specified by two $n \times n$ matrices; so we care about algorithms that run in time polynomial in n. [4]

[4]Other desiderata: e.g. "decentralised" style of algorithm

[5]Gilboa and Zemel: Nash and Correlated Equilibria: Some Complexity Considerations, *GEB* '89. Conitzer and Sandholm: Complexity Results about Nash Equilibria, *IJCAI* '03

NP-Completeness of finding "good" Nash equilibria

2-player game: specified by two $n \times n$ matrices; so we care about algorithms that run in time polynomial in n. [4]

It is **NP**-hard to find (for 2-player games) the NE with highest social welfare.[5] CS'03 paper gives a class of games for which various restricted NE are hard to find, e.g. NE that guarantees player 1 a payoff of α.

---

[4] Other desiderata: e.g. "decentralised" style of algorithm

[5] Gilboa and Zemel: Nash and Correlated Equilibria: Some Complexity Considerations, *GEB* '89. Conitzer and Sandholm: Complexity Results about Nash Equilibria, *IJCAI* '03

NP-Completeness of finding "good" Nash equilibria

2-player game: specified by two $n \times n$ matrices; so we care about algorithms that run in time polynomial in n. [4]

It is **NP**-hard to find (for 2-player games) the NE with highest social welfare.[5] CS'03 paper gives a class of games for which various restricted NE are hard to find, e.g. NE that guarantees player 1 a payoff of α.

The following is a brief sketch of their construction (note: after this, I will give 2 simpler reductions in detail)

---

[4] Other desiderata: e.g. "decentralised" style of algorithm

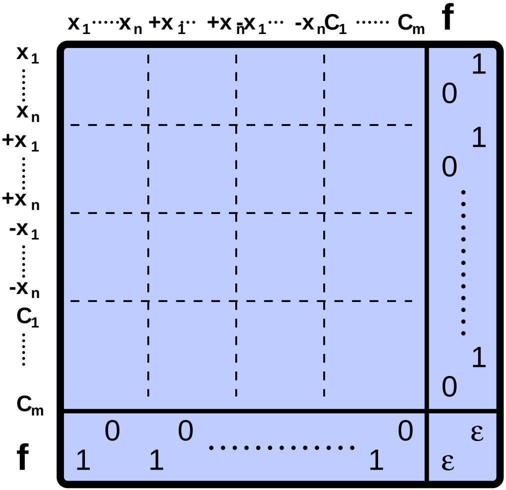
[5] Gilboa and Zemel: Nash and Correlated Equilibria: Some Complexity Considerations, *GEB* '89. Conitzer and Sandholm: Complexity Results about Nash Equilibria, *IJCAI* '03

NP-Completeness of finding "good" Nash equilibria

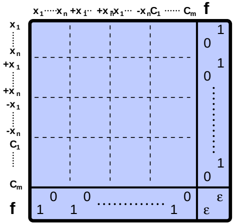
Reduce from SATISFIABILITY: Given a CNF formula Φ with n variables and m clauses, find a satisfying assignment

Construct game $\mathcal{G}_\Phi$ having $3n + m + 1$ actions per player (hence of size polynomial in Φ)

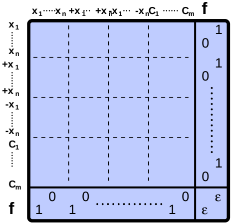
NP-Completeness of finding "good" Nash equilibria



NP-Completeness of finding “good” Nash equilibria



- (f, f) is a Nash equilibrium.



- (f, f) is a Nash equilibrium.

Various other payoffs between 0 and n apply when neither player plays f. They are chosen such that

- if Φ is satisfiable, so also is a uniform distribution over a satisfying set of literals.
- No other Nash equilibria!

NP-Completeness of finding "good" Nash equilibria

Comment: This shows it is hard to find "best" NE, but clearly (f, f) is always easy to find.

NP-Completeness of finding "good" Nash equilibria

Comment: This shows it is hard to find "best" NE, but clearly (f, f) is always easy to find.

Should we expect it to be **NP**-hard to find *unrestricted* NE?

NP-Completeness of finding "good" Nash equilibria

Comment: This shows it is hard to find "best" NE, but clearly (f, f) is always easy to find.

Should we expect it to be **NP**-hard to find *unrestricted* NE?

General agenda of next part is to explain why we believe this is still hard, but not **NP**-hard.

Reduction between 2 versions of search for unrestricted NE: A simple example

zero-sum game (e.g. rock-paper-scissors): total payoff of all the players is constant. 2-player 0-sum games can be solved by LP (easy; later) unlike general 2-player games.

Reduction between 2 versions of search for unrestricted NE: A simple example

zero-sum game (e.g. rock-paper-scissors): total payoff of all the players is constant. 2-player 0-sum games can be solved by LP (easy; later) unlike general 2-player games.

Simple theorem

3-player zero-sum games are at least as hard as 2-player games.

Reduction between 2 versions of search for unrestricted NE: A simple example

zero-sum game (e.g. rock-paper-scissors): total payoff of all the players is constant. 2-player 0-sum games can be solved by LP (easy; later) unlike general 2-player games.

Simple theorem

3-player zero-sum games are at least as hard as 2-player games.

To see this, take any $n \times n$ 2-player game $\mathcal{G}$.
Now add player 3 to $\mathcal{G}$, who is "passive" — he has just one action, which does not affect players 1 and 2, and player 3's payoff is the negation of the total payoffs of players 1 and 2.

Reduction between 2 versions of search for unrestricted NE: A simple example

zero-sum game (e.g. rock-paper-scissors): total payoff of all the players is constant. 2-player 0-sum games can be solved by LP (easy; later) unlike general 2-player games.

Simple theorem

3-player zero-sum games are at least as hard as 2-player games.

To see this, take any $n \times n$ 2-player game $\mathcal{G}$.
Now add player 3 to $\mathcal{G}$, who is "passive" — he has just one action, which does not affect players 1 and 2, and player 3's payoff is the negation of the total payoffs of players 1 and 2. So, players 1 and 2 behave as they did before, and player 3 just has the effect of making the game zero-sum. Any Nash equilibrium of this 3-player game is, for players 1 and 2, a NE of the original 2-player game.

Reduction: 2-player to symmetric 2-player

A symmetric game is one where "all players are the same": they all have the same set of actions, payoffs do not depend on a player's identity, only on actions chosen.
For 2-player games, this means the matrix diagrams (of the kind we use here) should be symmetric (as in fact they are in the examples we saw earlier).

A slightly more interesting theorem

symmetric 2-player games are as hard as general 2-player games.

Reduction: 2-player to symmetric 2-player

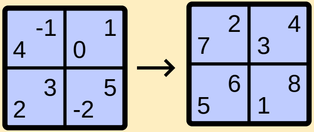
Given a $n \times n$ game $\mathcal{G}$, construct a symmetric $2n \times 2n$ game $\mathcal{G}' = f(\mathcal{G})$, such that given any Nash equilibrium of $\mathcal{G}'$ we can efficiently reconstruct a NE of $\mathcal{G}$.

Reduction: 2-player to symmetric 2-player

Given a $n \times n$ game $\mathcal{G}$, construct a symmetric $2n \times 2n$ game $\mathcal{G}' = f(\mathcal{G})$, such that given any Nash equilibrium of $\mathcal{G}'$ we can efficiently reconstruct a NE of $\mathcal{G}$.

First step: if any payoffs in $\mathcal{G}$ are negative, add a constant to *all* payoffs to make them all positive.

Example:



Nash equilibria are unchanged by this (game is "strategically equivalent")

Reduction: 2-player to symmetric 2-player

So now let's assume $\mathcal{G}$'s payoffs are all positive. Next stage:

$$\mathcal{G}' = \begin{pmatrix} 0 & \mathcal{G} \\ \mathcal{G}^T & 0 \end{pmatrix}$$

Example:

7, 2	3, 4
5, 6	1, 8

→

0, 0	0, 0	7, 2	3, 4
0, 0	0, 0	5, 6	1, 8
2, 7	6, 5	0, 0	0, 0
4, 3	8, 1	0, 0	0, 0

Reduction: 2-player to symmetric 2-player

Now suppose we solve the $2n \times 2n$ game $\mathcal{G}' = \begin{pmatrix} 0 & \mathcal{G} \\ \mathcal{G}^T & 0 \end{pmatrix}$

Let p and q denote the probabilities that players 1 and 2 use their first n actions, in some given solution.

$$\begin{array}{cc} & \begin{array}{cc} q & 1-q \end{array} \\ \begin{array}{c} p \\ 1-p \end{array} & \begin{pmatrix} 0 & \mathcal{G} \\ \mathcal{G}^T & 0 \end{pmatrix} \end{array}$$

If $p = q = 1$, both players receive payoff 0, and both have incentive to change their behavior, by assumption that $\mathcal{G}$'s payoffs are all positive. (and similarly if $p = q = 0$).

So we have $p > 0$ and $1 - q > 0$, or alternatively, $1 - p > 0$ and $q > 0$.

Assume $p > 0$ and $1 - q > 0$ (the analysis for the other case is similar).

Reduction: 2-player to symmetric 2-player

Let $\{p_1, ..., p_n\}$ be the probabilities used by player 1 for his first n actions, $\{q_1, \ldots q_n\}$ the probs for player 2's second n actions.

$$\begin{array}{cc} & q \quad (q_1 ... q_n) \\ (p_1, ... p_n) & \\ 1-p & \end{array} \begin{pmatrix} 0 & \mathcal{G} \\ \mathcal{G}^T & 0 \end{pmatrix}$$

Note that $p_1 + \ldots + p_n = p$ and $q_1 + \ldots + q_n = 1 - q$.

Reduction: 2-player to symmetric 2-player

Let $\{p_1, ..., p_n\}$ be the probabilities used by player 1 for his first n actions, $\{q_1, \ldots q_n\}$ the probs for player 2's second n actions.

$$
\begin{array}{rc}
 & q \quad (q_1 ... q_n) \\
(p_1, ... p_n) & \begin{pmatrix} 0 & \mathcal{G} \\ \mathcal{G}^T & 0 \end{pmatrix} \\
1 - p &
\end{array}
$$

Note that $p_1 + \ldots + p_n = p$ and $q_1 + \ldots + q_n = 1 - q$.

Then $(p_1/p, \ldots, p_n/p)$ and $(q_1/(1-q), \ldots, q_n/(1-q))$ are a Nash equilibrium of $\mathcal{G}$!
To see this, consider the diagram; they form a best response to each other for the top-right part.

Road-map of where we're going

- I pointed out (without proof) that NASH is a total search problem
- In fact, it's a **NP** total search problem
- We can relate variants of NASH, via reductions

Next:

- Let's make sure we understand the different between typical **NP** search problem, and **NP** total search problem
- We'll see that it would be hard to relate the two
- We can sometimes relate various **NP** total search problems (easier to "compare like with like")

NP Search Problems

NP decision problems: answer yes/no to questions that belong to some class. e.g. SATISFIABILITY: questions of the form Is boolean formula Φ satisfiable?

NP Search Problems

NP decision problems: answer yes/no to questions that belong to some class. e.g. SATISFIABILITY: questions of the form Is boolean formula Φ satisfiable?
Given the question Is formula Φ satisfiable? there is a fundamental asymmetry between answering yes and no.

NP Search Problems

NP decision problems: answer yes/no to questions that belong to some class. e.g. SATISFIABILITY: questions of the form Is boolean formula Φ satisfiable?

Given the question Is formula Φ satisfiable? there is a fundamental asymmetry between answering yes and no.

If yes, there exists a small "certificate" that the answer is yes, namely a satisfying assignment. A certificate consists of information that allows us to check (in poly time) that the answer is yes.

NP Search Problems

NP decision problems: answer yes/no to questions that belong to some class. e.g. SATISFIABILITY: questions of the form Is boolean formula Φ satisfiable?

Given the question Is formula Φ satisfiable? there is a fundamental asymmetry between answering yes and no.

If yes, there exists a small "certificate" that the answer is yes, namely a satisfying assignment. A certificate consists of information that allows us to check (in poly time) that the answer is yes.

A **NP** decision problem has a corresponding *search problem*: e.g. given Φ, find $\mathbf{x}$ such that $\Phi(\mathbf{x}) = true$ (or say "no" if Φ is not satisfiable.)

Example of *Total* search problem in **NP**

FACTORING

Input	number N
Output	prime factorisation of N

[6] polynomial in the number of digits in N

Example of *Total* search problem in **NP**

FACTORING

Input number N
Output prime factorisation of N

e.g. Input 50 should result in output $2 \times 5 \times 5$.

[6] polynomial in the number of digits in N

Example of *Total* search problem in **NP**

Factoring

Input	number N
Output	prime factorisation of N

e.g. Input 50 should result in output $2 \times 5 \times 5$.
Given output $N = N_1 \times N_2 \times \ldots N_p$, it can be checked in polynomial time[6] that the numbers $N_1, \ldots, N_p$ are prime, and their product is N.

[6] polynomial in the number of digits in N

Example of *Total* search problem in **NP**

FACTORING

Input	number N
Output	prime factorisation of N

e.g. Input 50 should result in output $2 \times 5 \times 5$.

Given output $N = N_1 \times N_2 \times \ldots N_p$, it can be checked in polynomial time[6] that the numbers $N_1, \ldots, N_p$ are prime, and their product is N.

Hence, FACTORING is in **FNP**. But, it's a total search problem — every number has a prime factorization.

[6] polynomial in the number of digits in N

Example of *Total* search problem in **NP**

FACTORING

Input number N
Output prime factorisation of N

e.g. Input 50 should result in output $2 \times 5 \times 5$.
Given output $N = N_1 \times N_2 \times \dots N_p$, it can be checked in polynomial time[6] that the numbers $N_1, \dots, N_p$ are prime, and their product is N.
Hence, FACTORING is in **FNP**. But, it's a total search problem — every number has a prime factorization.
It also seems to be hard! Cryptographic protocols use the belief that it is intrinsically hard. But probably not **NP**-complete

[6] polynomial in the number of digits in N

Another **NP** total search problem

EQUAL-SUBSETS

Input positive integers $a_1, \ldots, a_n$; $\Sigma_i a_i < 2^n - 1$

Output Two distinct subsets of these numbers that add up to the same total

Another **NP** total search problem

Equal-subsets

Input positive integers $a_1, \ldots, a_n$; $\Sigma_i a_i < 2^n - 1$

Output Two distinct subsets of these numbers that add up to the same total

Example:

$$42, 5, 90, 98, 99, 100, 64, 70, 78, 51$$

Another **NP** total search problem

Equal-subsets

Input positive integers $a_1, \ldots, a_n$; $\Sigma_i a_i < 2^n - 1$

Output Two distinct subsets of these numbers that add up to the same total

Example:

$$42, 5, 90, 98, 99, 100, 64, 70, 78, 51$$

Solutions include $42 + 78 + 100 = 51 + 70 + 99$ and $42 + 5 + 51 = 98$.

Another **NP** total search problem

EQUAL-SUBSETS

Input positive integers $a_1, \ldots, a_n$; $\Sigma_i a_i < 2^n - 1$

Output Two distinct subsets of these numbers that add up to the same total

Example:

$$42, 5, 90, 98, 99, 100, 64, 70, 78, 51$$

Solutions include $42 + 78 + 100 = 51 + 70 + 99$ and $42 + 5 + 51 = 98$.

EQUAL-SUBSETS $\in$ **NP** (usual "guess and test" approach). But it is not known how to find solutions in polynomial time. The problem looks a bit like the **NP**-complete problem SUBSET SUM.

So, should we expect EQUAL-SUBSETS to be **NP**-hard?

So, should we expect EQUAL-SUBSETS to be **NP**-hard?

No we should not [Megiddo (1988)] (The following is important. Also works for FACTORING etc.)

If any total search problem (e.g. EQUAL-SUBSETS) is **NP**-complete, then it follows that **NP**=**co-NP**, which is generally believed not to be the case.

So, should we expect EQUAL-SUBSETS to be **NP**-hard?

No we should not [Megiddo (1988)] (The following is important. Also works for FACTORING etc.)

If any total search problem (e.g. EQUAL-SUBSETS) is **NP**-complete, then it follows that **NP**=**co-NP**, which is generally believed not to be the case.
To see why, suppose it is **NP**-complete, thus
SAT $\leq_p$ EQUAL-SUBSETS.
Then there is an algorithm $\mathcal{A}$ for SAT that runs in polynomial time, provided that it has access to poly-time algorithm $\mathcal{A}'$ for EQUAL SUBSETS.
Now suppose $\mathcal{A}$ is given a *non-satisfiable* formula Φ. Presumably it calls $\mathcal{A}'$ some number of times, and receives a sequence of solutions to various instances of EQUAL SUBSETS, and eventually the algorithm returns the answer "no, Φ is not satisfiable".

So, should we expect EQUAL SUBSETS to be **NP**-hard?

Now suppose that we replace $\mathcal{A}'$ with the natural "guess and test" non-deterministic algorithm for EQUAL-SUBSETS.
We get a non-deterministic polynomial-time algorithm for SAT.
Notice that when Φ is given to this new algorithm, the "guess and test" subroutine for EQUAL SUBSETS can produce the same sequence of solutions to the instances it receives, and as a result, the entire algorithm can recognize this non-satisfiable formula Φ as before. Thus we have **NP** algorithm that recognizes unsatisfiable formulae, which gives the consequence **NP**=**co-NP**.

Classes of total search problems

TFNP: total function problems in **NP**. We want to understanding the difficulty of certain **TFNP** problems.

NASH and EQUAL-SUBSETS do not seem to belong to **P** but are probably not **NP**-complete, due to being total search problems.

Papadimitriou (1991,4) introduced a number of classes of total search problems.

Classes of total search problems

TFNP: total function problems in **NP**. We want to understanding the difficulty of certain **TFNP** problems.
NASH and EQUAL-SUBSETS do not seem to belong to **P** but are probably not **NP**-complete, due to being total search problems.
Papadimitriou (1991,4) introduced a number of classes of total search problems.

General observation:

"$X \in$ **TFNP**" doesn't say *why* X is total. But...
syntactic sub-classes of **TFNP** contain problems whose totality is due to some combinatorial principle. (there's a non-constructive existence proof with hard-to-compute step)

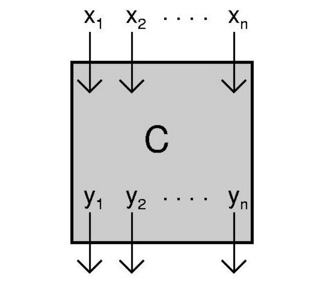
PPP stands for "polynomial pigeonhole principle"; used to prove that EQUAL-SUBSETS is a total search problem.
"A function whose domain is larger than its range has 2 inputs with the same output"

The generic **PPP** problem

Definition:

Pigeonhole circuit is the following search problem:

Input: boolean circuit C, n inputs, n outputs
Output: A boolean vector $\mathbf{x}$ such that $C(\mathbf{x}) = \mathbf{0}$, or alternatively, vectors $\mathbf{x}$ and $\mathbf{x}'$ such that $C(\mathbf{x}) = C(\mathbf{x}')$.



The "most general" computational total search problem for which the pigeonhole principle guarantees an efficiently checkable solution.

Various equivalent definitions of PIGEONHOLE CIRCUIT

With regard to questions of polynomial time computation, the following are equivalent

- n inputs/outputs; C of size n^2
- Let p be a polynomial; n inputs/outputs, C of size $p(n)$
- n is number of gates in C, number of inputs $=$ number of outputs.

Proof of equivalences via reductions: If version i is in **P** then version j is in **P**.

The complexity class **PPP**

Definition

A problem X belongs to **PPP** if X reduces to PIGEONHOLE CIRCUIT (in poly time).
Problem X is **PPP**-complete is in addition, PIGEONHOLE CIRCUIT reduces to X.

The complexity class **PPP**

Definition

A problem X belongs to **PPP** if X reduces to PIGEONHOLE CIRCUIT (in poly time).
Problem X is **PPP**-complete is in addition, PIGEONHOLE CIRCUIT reduces to X.

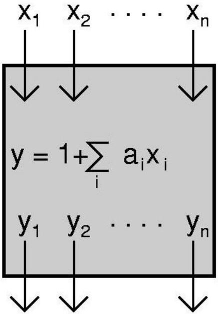
Analogy

Thus, **PPP** is to PIGEONHOLE CIRCUIT as **NP** is to SATISFIABILITY (or CIRCUIT SAT, or any other **NP**-complete problem).

PIGEONHOLE CIRCUIT seems to be hard (it looks like CIRCUIT SAT) but (recall) probably not **NP**-hard.

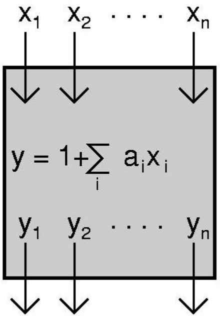
What we know about Equal-subsets

Equal-subsets belongs to **PPP**...



What we know about EQUAL-SUBSETS

EQUAL-SUBSETS belongs to **PPP**...
but it is not known whether it is complete for **PPP**. (this is unsatisfying.)



Subclasses of **PPP**

Problem with **PPP**: no interesting **PPP**-completeness results.
PPP fails to "capture the complexity" of apparently hard problems, such as Nash.
Here is a specialisation of the pigeonhole principle:

"Suppose directed graph G has indegree and outdegree at most 1. Given a source, there must be a sink."

Subclasses of **PPP**

Problem with **PPP**: no interesting **PPP**-completeness results.
PPP fails to "capture the complexity" of apparently hard problems, such as NASH.
Here is a specialisation of the pigeonhole principle:

"Suppose directed graph G has indegree and outdegree at most 1. Given a source, there must be a sink."

Why is this the pigeonhole principle?
$G = (V, E)$; $f : V \to V$ defined as follows:
For all $e = (u, v)$, let $f(u) = v$. If u is a sink, let $f(u) = u$.
Let $s \in E$ be a source. So $s \notin range(f)$. The pigeonhole principle says that 2 vertices must be mapped by f to the same vertex.

Subclasses of **PPP**

$G = (V, E)$, $V = \{0, 1\}^n$.
G is represented using 2 circuits P and S ("predecessor" and "successor") with n inputs/outputs.
G has 2^n vertices (bit strings); $\mathbf{0}$ is source. $(\mathbf{x}, \mathbf{x}')$ is an edge iff $\mathbf{x}' = S(\mathbf{x})$ and $\mathbf{x} = P(\mathbf{x}')$.
Thus, G is a BIG graph and it's not clear how best to find a sink, even though you know it's there!

Subclasses of **PPP**

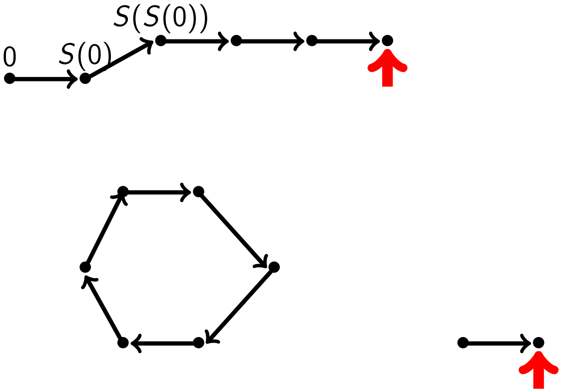
$G = (V, E)$, $V = \{0, 1\}^n$.
G is represented using 2 circuits P and S ("predecessor" and "successor") with n inputs/outputs.
G has 2^n vertices (bit strings); $\mathbf{0}$ is source. $(\mathbf{x}, \mathbf{x}')$ is an edge iff $\mathbf{x}' = S(\mathbf{x})$ and $\mathbf{x} = P(\mathbf{x}')$.
Thus, G is a BIG graph and it's not clear how best to find a sink, even though you know it's there!

Definition: FIND A SINK

Input: (concisely represented) graph G, source $v \in G$
Output: $v' \in G$, v' is a sink

picture on next slide...

Search the graph for a sink



But, if you find a sink, it's easy to *check* it's genuine! So, search is in **FNP**.

Parity argument on a graph

A weaker version of the "there must be a sink":

"Suppose directed graph G has indegree and outdegree at most 1. Given a source, there must be another vertex that is either a source or a sink."
picture on next slide...

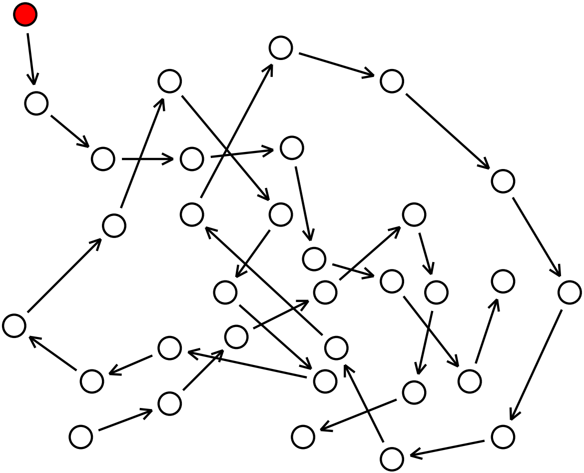
Definition: END OF LINE

Input: graph G, source $v \in G$
Output: $v' \in G$, $v' \neq v$ is either a source or a sink

PPAD is defined in terms of END OF LINE the same way that **PPP** is defined in terms of PIGEONHOLE CIRCUIT.

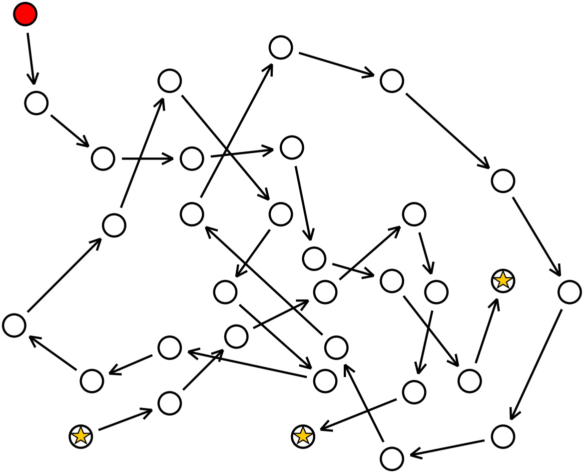
Equivalent (more general-looking) formulation: If G (not necessarily of in/out-degree 1) has an "unbalanced vertex", then it must have another one. "parity argument on a directed graph"

END OF LINE graph



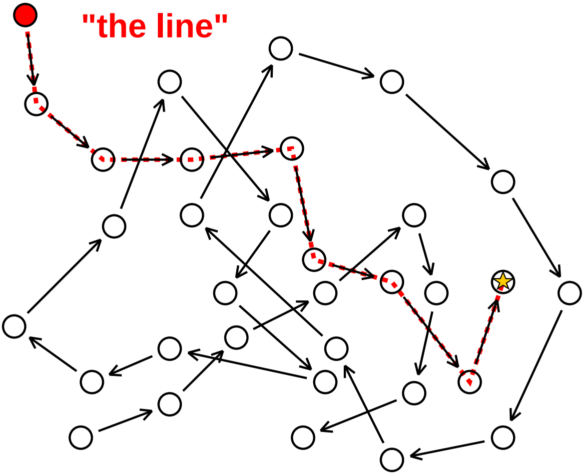
You are given a node with degree 1 (colored red here)

END 0F LINE graph



The highlighted nodes are **PPAD**-complete to find...
(NOTE: odd number of solutions!)

END OF LINE graph



The one attached to the red node is **PSPACE**-complete to find!

Digression on **PSPACE**-completeness

Given a graph G (presented as circuits S and P) with source $\mathbf{0}$, there exists a sink $\mathbf{x}$ such that $\mathbf{x} = S(S(\ldots(S(0))\ldots))$.

It's total search problem, but completely different; note the solution has no (obvious) certificate...

PSPACE-complete — the search for this $\mathbf{x}$ is computationally equivalent to search for the final configuration of a polynomially space-bounded Turing machine.[7]

Nash equilibria computed by the Lemke-Howson algorithm are also **PSPACE**-complete to compute[8] "paradox" since L-H is "efficient in practice"

[7] Papadimitriou: On the complexity of the parity argument and other inefficient proofs of existence. *JCSS* '94; Crescenzi & Papadimitriou: Reversible Simulation of Space-Bounded Computations. *TCS* '95

[8] G, Papadimitriou, Savani: The Complexity of the Homotopy Method, Equilibrium Selection, and Lemke-Howson Solutions. *FOCS* '11

Subclasses of **PPP**

- **PPADS** is the complexity class defined w.r.t. Find a sink (i.e. problems reducible to Find a sink)
- **PPAD**: problems reducible to End of line.

$$\textbf{PPAD} \subseteq \textbf{PPADS} \subseteq \textbf{PPP}$$

because

$$\text{End of line} \leq_p \text{Find a sink} \leq_p \text{Pigeonhole circuit.}$$

If we could e.g. reduce Find a sink back to End of line, then that would show that **PPAD** and **PPADS** are the same, but this has not been achieved...

Subclasses of **PPP**

- **PPADS** is the complexity class defined w.r.t. Find a sink (i.e. problems reducible to Find a sink)
- **PPAD**: problems reducible to End of line.

$$\textbf{PPAD} \subseteq \textbf{PPADS} \subseteq \textbf{PPP}$$

because

$$\textsc{End of line} \leq_p \textsc{Find a sink} \leq_p \textsc{Pigeonhole circuit}.$$

If we could e.g. reduce Find a sink back to End of line, then that would show that **PPAD** and **PPADS** are the same, but this has not been achieved...
In the mean time, it turns out that **PPAD** is the sub-class of **PPP** that captures the complexity of Nash and related problems.
PPAD turns out to give rise to "interesting" reductions

Finally, here is why we care about **PPAD**. It seems to capture the complexity of a number of problems where a solution is guaranteed by *Brouwer's fixed point Theorem*.

NASH is **PPAD**-complete

Finally, here is why we care about **PPAD**. It seems to capture the complexity of a number of problems where a solution is guaranteed by *Brouwer's fixed point Theorem*.

Two parts to the proof:

1. NASH is in **PPAD**, i.e. NASH $\leq_p$ END OF LINE
2. END OF LINE $\leq_p$ NASH

Reducing NASH to END OF LINE

We need to show NASH $\leq_p$ END OF LINE.
That is, we need two functions f and g such that given a game $\mathcal{G}$, $f(\mathcal{G}) = (P, S)$ where P and S are circuits that define an END OF LINE instance...
Given a solution $\mathbf{x}$ to (P, S), $g(\mathbf{x})$ is a solution to $\mathcal{G}$.

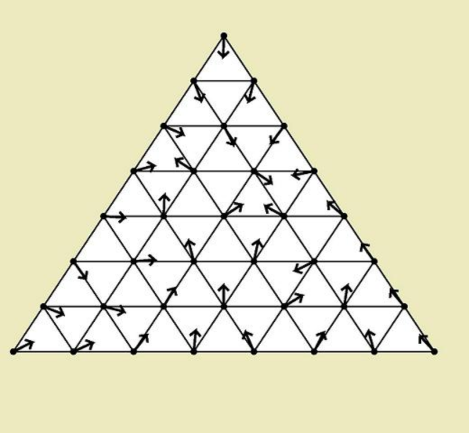
Notes

- NASH is taken to mean: find an *approximate* NE
- Reduction is a computational version of Nash's theorem
- Nash's theorem uses *Brouwer's fixed point theorem*, which in turn uses *Sperner's lemma*; the reduction shows how these results are proven...

Reducing NASH to END OF LINE

For a k-player game $\mathcal{G}$, solution space is compact domain $(\Delta_n)^k$
Given a candidate solution $(p_1^1, ...p_n^1, \ldots, p_1^k, ...p_n^k)$, a point in this compact domain, $f_{\mathcal{G}}$ displaces that point according to the *direction* that player(s) prefer to change their behavior.
$f_{\mathcal{G}}$ is a *Brouwer* function, a continuous function from a compact domain to itself.
Brouwer FPT: There exists $\mathbf{x}$ with $f_{\mathcal{G}}(\mathbf{x}) = \mathbf{x}$ — why?

Reduction to BROUWER



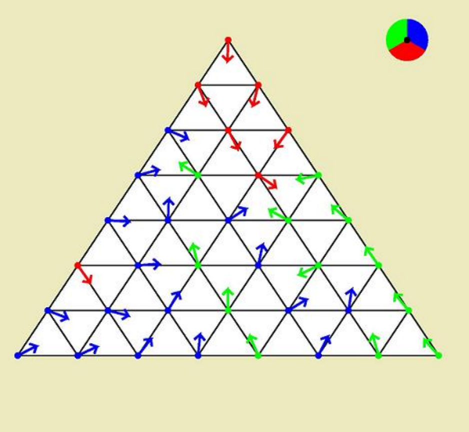
domain $(\Delta_n)^k$
divide into simplices of size ϵ/n

Arrows show direction of

Brouwer function, e.g. $f_{\mathcal{G}}$

If $f_{\mathcal{G}}$ is constructed sensibly, look for simplex where arrows go in all directions — *sufficient* condition for being near ϵ-NE.

Reduction to SPERNER

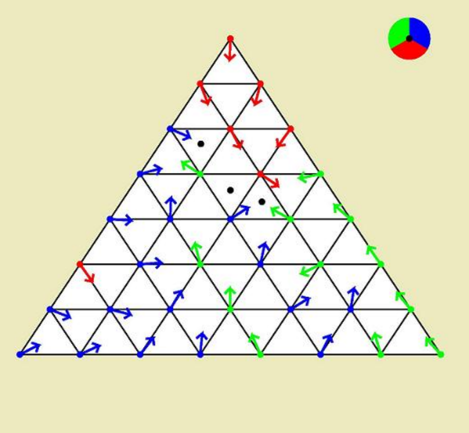


$(\Delta_n)^k$: polytope in R^{nk}; $nk + 1$ colors.

Color "grid points":

- red direction away from top;
- green away from bottom RH corner
- blue away from bottom LH corner

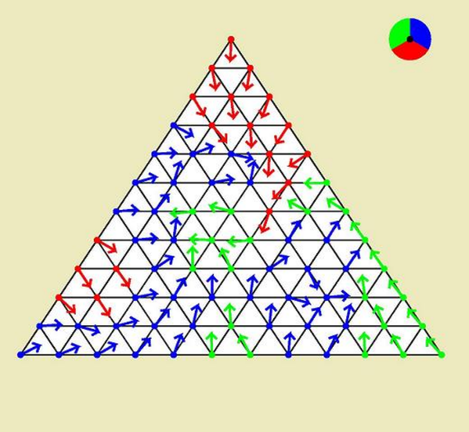
Reduction to SPERNER



Sperner's Lemma (in 2-D): promises "trichomatic triangle"

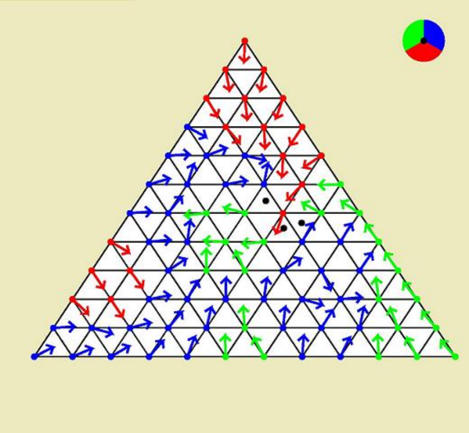
If so, trichromatic triangles at increasingly higher and higher resolutions should lead us to a Brouwer fixpoint...

Reduction to SPERNER



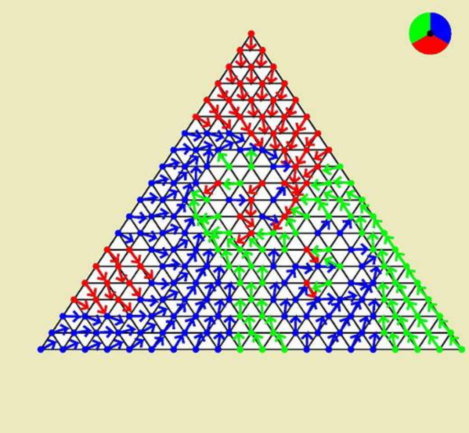
Let's try that out (and then we'll prove Sperner's lemma)

Reduction to SPERNER



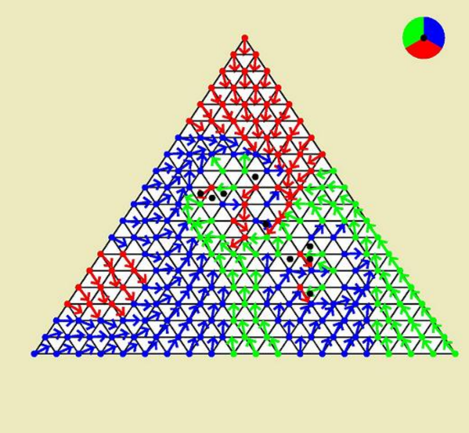
Black spots show the trichromatic triangles

Reduction to SPERNER



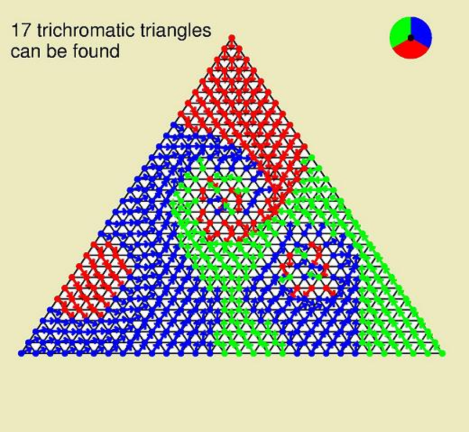
Higher-resolution version

Reduction to SPERNER



Again, black spots show trichromatic triangles

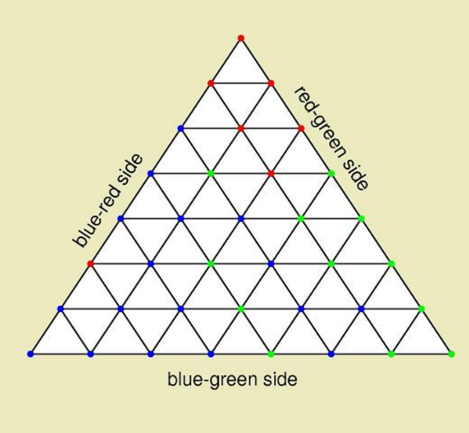
Reduction to Sperner



Once more — again we find trichromatic triangles!

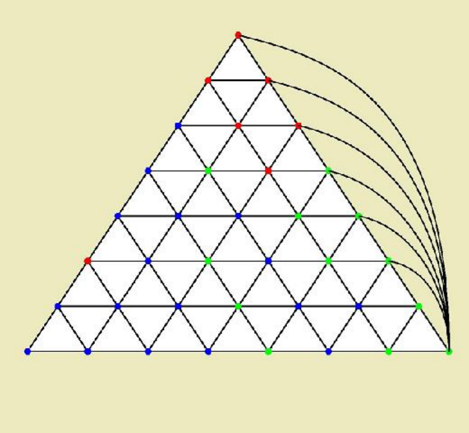
Next: convince ourselves they always can be found, for <u>any</u> Brouwer function.

Sperner's Lemma



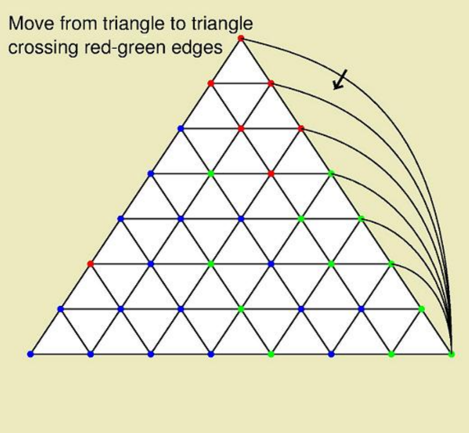
Suppose we color the grid points under the constraint shown in the diagram. Why can we be *sure* that there is a trichromatic triangle?

Reduction to SPERNER



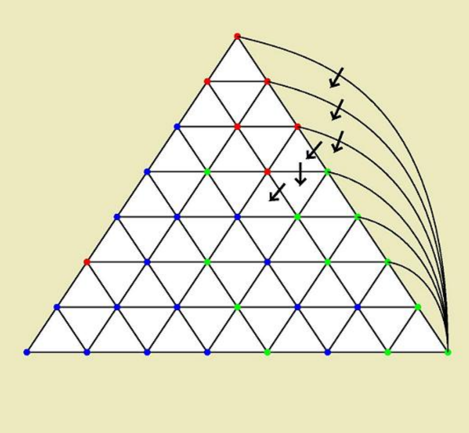
Add some edges such that only one red/green edge is open to the outside

Reduction to SPERNER



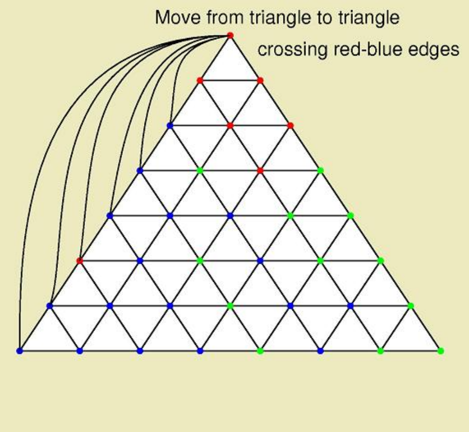
red/green edges are "doorways" that connect the triangles

Reduction to SPERNER



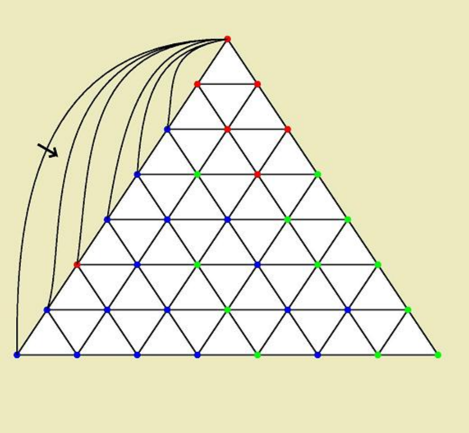
Keep going — we end up at a trichromatic triangle!

Reduction to SPERNER



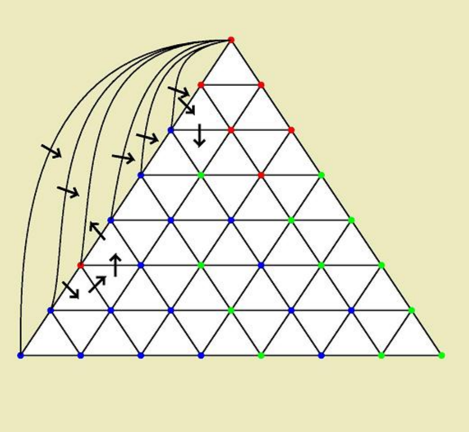
We can do the same trick w.r.t. the red/blue edges

Reduction to SPERNER



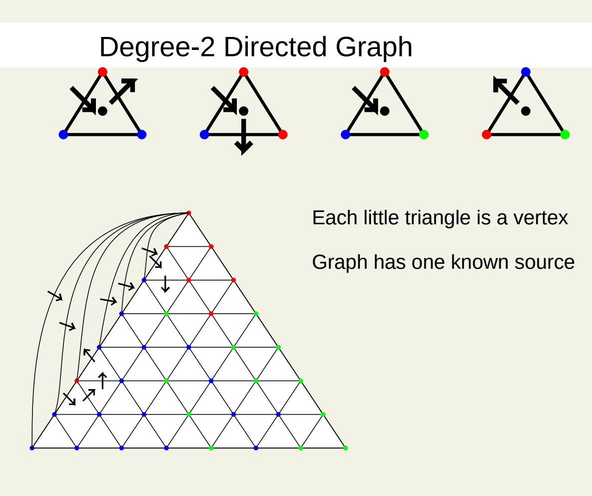
Now the red/blue edges are doorways

Reduction to SPERNER



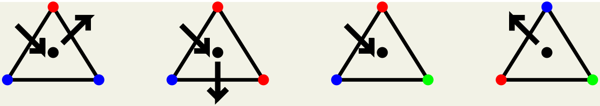
Keep going through them — eventually find a panchromatic triangle!

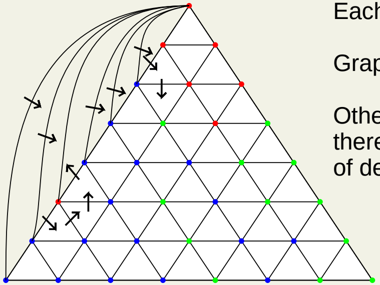
Reduction to SPERNER



Essentially, Sperner's lemma converts the function into an END OF LINE graph!

Degree-2 Directed Graph





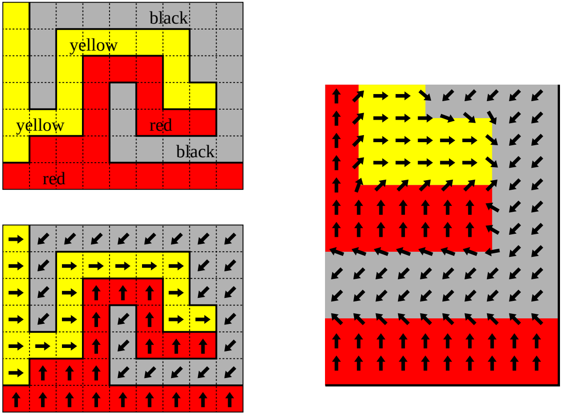
Each little triangle is a vertex

Graph has one known source

Other than the known source, there must be an odd number of degree-1 vertices.

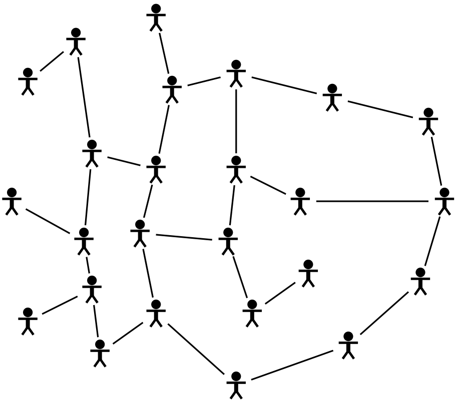
Reducing END OF LINE to NASH

- END OF LINE $\leq_p$ BROUWER
- BROUWER $\leq_p$ GRAPHICAL NASH
- GRAPHICAL NASH $\leq_p$ NASH



trichromatic point corresponds to fixpoint

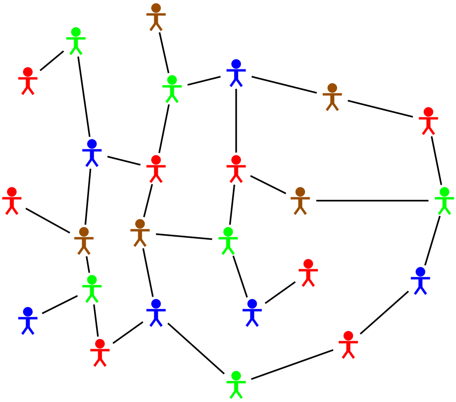
Graphical games



Players $1, \ldots, n$
Players: nodes of graph G of low degree d
strategies $1, \ldots, t$
utility depends on strategies in neighbourhood
$n.t^{(d+1)}$ numbers describe game

Compact representation of game with many players.

Graphical Nash $\leq_p$ Nash



Color the graph s.t.

- proper coloring
- each vertex's neighbors get distinct colors

Normal-form game:

- one "super-player" for each color
- Each super-player simulates entire set of players having that color

Naive bound of $d^2 + 1$ on number of colors needed

So we have a small number of super-players (given that d is small).
Problem: If blue super-player chooses an action for each member of his "team" he has t^n possible actions — can't write that down in normal form!

So we have a small number of super-players (given that d is small).
Problem: If blue super-player chooses an action for each member of his "team" he has t^n possible actions — can't write that down in normal form!
Solution: Instead, he will just choose one member v of his team at random, and choose an action for v, just $t.n$ possible actions!

Graphical Nash $\leq_p$ Nash

So we have a small number of super-players (given that d is small).
Problem: If blue super-player chooses an action for each member of his "team" he has t^n possible actions — can't write that down in normal form!
Solution: Instead, he will just choose one member v of his team at random, and choose an action for v, just $t.n$ possible actions!
so what we have to do is: Incentivize each super-player to pick a random team member v; and further, incentivize him to pick a best response for v afterwards
This is done by choice of payoffs to super-players (in our graph, $\{red, blue, green, brown\}$)

Graphical Nash $\leq_p$ Nash

If we have coloring $\{red, blue, green, brown\}$
The actions of the *red* super-player are of the form: Choose a red vertex on the graph, then choose an action in $\{1, ..., s\}$.
Payoffs:

- If I choose a node v, and the other super-players choose nodes in v's neighborhood, then red gets the payoff that v would receive
- Also, if red chooses the i-th red vertex (in some given ordering) and blue chooses his i-th vertex, then red receives (big) payoff M and blue gets penalty $-M$ (and simialrly for other pairs of super-players)

The 2nd of these means a super-player will randomize amongst nodes of his color in G. The first means that when he his chosen $v \in G$, his choice of v's action should be a best response.

Why we needed a proper colouring:
Because when a super-player chooses v, there should be some positive probability that v's neighbors get chosen; AND these choices should be made independently.

Next: the quest for positive results: poly-time algorithms for approximate equilibria

Approximate Nash equilibria

Hardness results apply to $\epsilon = 1/n$; generally $\epsilon = 1/p(n)$ for polynomial p. No FPTAS; main open problem is possible existence of a PTAS. Failing that, better constant approximations would be nice

What if e.g. $\epsilon = 1/3$?

- 2 players - let R and C be matrices of row/column players's utils
- let x and y denote the row and column players' strategies; let e_i be vector with 1 in component i, zero elsewhere.
- For all i, $x^{\mathrm{T}} R y \geq e_i^{\mathrm{T}} R y - \epsilon$.
- For all j, $x^{\mathrm{T}} C y \geq x^{\mathrm{T}} C e_j - \epsilon$.
- Remember: payoffs are re-scaled into $[0, 1]$.

Zero-sum games are in **P**

Zero-sum games: $C = -R$.

Player 1: $\min_x \max_y(-xRy)$
$-xRy$ is player 2's payoff
Equivalently: $\min_x \max_j(-xRe_j)$
Player 2's best response can be achieved by a pure strategy

LP:

minimise v_2 subject to the constraints

- $x \geq \mathbf{0}_n$; $x^{\mathrm{T}}\mathbf{1}_n = 1$
- $y \geq \mathbf{0}_n$; $y^{\mathrm{T}}\mathbf{1}_n = 1$
- for all j, $v_2 \geq -x^{\mathrm{T}}Re_j$

A simple algorithm (no LP required)

Guarantee $\epsilon = \frac{1}{2}$ [9]

$\frac{1}{2}$

0.2 / 0	0.9 / 0.1	0.2 / 0.2
0.2 / 0.3	0.1 / 0.4	0.2 / 0.5
0.2 / 0.6	0.2 / 0.7	0.8 / 0.8

1. Player 1 chooses arbitrary strategy i; gives it probability $\frac{1}{2}$.

[9] Daskalakis, Mehta and Papadimitriou: A note on approximate Nash equilibria, *WINE*'06, *TCS*'09

A simple algorithm (no LP required)

Guarantee $\epsilon = \frac{1}{2}$ [9]

		1	
$\frac{1}{2}$	0.2 0	0.9 0.1	0.2 0.2
	0.2 0.3	0.1 0.4	0.2 0.5
	0.2 0.6	0.2 0.7	0.8 0.8

1. Player 1 chooses arbitrary strategy i; gives it probability $\frac{1}{2}$.
2. Player 1 chooses best response j; gives it probability 1.

[9] Daskalakis, Mehta and Papadimitriou: A note on approximate Nash equilibria, *WINE*'06, *TCS*'09

A simple algorithm (no LP required)

Guarantee $\epsilon = \frac{1}{2}$ [9]

		1	
$\frac{1}{2}$	0.2 0	0.9 0.1	0.2 0.2
	0.2 0.3	0.1 0.4	0.2 0.5
$\frac{1}{2}$	0.2 0.6	0.2 0.7	0.8 0.8

1. Player 1 chooses arbitrary strategy i; gives it probability $\frac{1}{2}$.
2. Player 1 chooses best response j; gives it probability 1.
3. Player 1 chooses best response to j; gives it probability $\frac{1}{2}$.

[9] Daskalakis, Mehta and Papadimitriou: A note on approximate Nash equilibria, *WINE*'06, *TCS*'09

How to find approximate solutions with $\epsilon < \frac{1}{2}$?

That was too easy...

How to find approximate solutions with $\epsilon < \frac{1}{2}$?

That was too easy...

But... next we will see that an algorithm for $\epsilon < \frac{1}{2}$ may need to find mixed strategies having more than a constant support size.

The *support* of a probability distribution is the set of events that get non-zero probability — for a mixed strategy, all the pure strategies that may get chosen. In the previous algorithm, player 1's mixed strategy had support ≤ 2 and player 2's had support 1.

Consider random zero-sum win-lose games of size $n \times n$:[10]

0 / 1	0 / 1	1 / 0	1 / 0	0 / 1	0 / 1
1 / 0	0 / 1	0 / 1	0 / 1	1 / 0	1 / 0
0 / 1	1 / 0	1 / 0	1 / 0	0 / 1	1 / 0
1 / 0	1 / 0	0 / 1	0 / 1	1 / 0	0 / 1
1 / 0	0 / 1	1 / 0	0 / 1	0 / 1	1 / 0
0 / 1	1 / 0	1 / 0	0 / 1	1 / 0	1 / 0

[10] Feder, Nazerzadeh and Saberi: Approximating Nash Equilibria using Small-Support Strategies, *ACM-EC*'07

Consider random zero-sum win-lose games of size $n \times n$:[10]

			1			
1	1, 0	1, 0	0, 1	0, 1	1, 0	1, 0
	0, 1	1, 0	1, 0	1, 0	0, 1	0, 1
	1, 0	0, 1	0, 1	0, 1	1, 0	0, 1
	0, 1	0, 1	1, 0	1, 0	0, 1	1, 0
	0, 1	1, 0	0, 1	1, 0	1, 0	0, 1
	1, 0	0, 1	0, 1	1, 0	0, 1	0, 1

1. With high probability, for any pure strategy by player 1, player 2 can "win"

[10] Feder, Nazerzadeh and Saberi: Approximating Nash Equilibria using Small-Support Strategies, *ACM-EC*'07

more than constant support size for $\epsilon < \frac{1}{2}$:

Consider random zero-sum win-lose games of size $n \times n$:[10]

					1	
	1, 0	1, 0	0, 1	0, 1	1, 0	1, 0
0.4	0, 1	1, 0	1, 0	1, 0	0, 1	0, 1
	1, 0	0, 1	0, 1	0, 1	1, 0	0, 1
0.6	0, 1	0, 1	1, 0	1, 0	0, 1	1, 0
	0, 1	1, 0	0, 1	1, 0	1, 0	0, 1
	1, 0	0, 1	0, 1	1, 0	0, 1	0, 1

1. With high probability, for any pure strategy by player 1, player 2 can "win"
2. Indeed, as n increases, this is true if player 1 may mix 2 of his strategies

[10] Feder, Nazerzadeh and Saberi: Approximating Nash Equilibria using Small-Support Strategies, *ACM-EC*'07

$1/n$	0 1	0 1	1 0	1 0	0 1	0 1
$1/n$	1 0	0 1	0 1	0 1	1 0	1 0
$1/n$	0 1	1 0	1 0	1 0	0 1	1 0
$1/n$	1 0	1 0	0 1	0 1	1 0	0 1
$1/n$	1 0	0 1	1 0	0 1	0 1	1 0
$1/n$	0 1	1 0	1 0	0 1	1 0	1 0

1. But, for large n, player 1 can guarantee a payoff of about $1/2$ by randomizing over his strategies (w.h.p., as n increases)
2. Given any constant support size κ, there is n large enough such that the other player can win against any mixed strategy that uses κ pure strategies. So, small-support strategies are $1/2$ worse than the fully-mixed strategy.

How big a support do you need?

$O(\log(n))$ is also an upper bound (for any constant ϵ) [11]

[11] Althofer 1994: On sparse approximations to randomized strategies and convex combinations *Linear algebra and its applecations* 1994; Lipton, Markakis, & Mehta: Playing Large Games using Simple Strategies. (extension from 2-player case to k-player case)

$O(\log(n))$ is also an upper bound (for any constant ϵ) [11]

How to prove the above –

Define an "empirical NE" as: draw N samples from Nash equilibrium x and y; replace x, y with resulting empirical distributions $\bar{x}$ and $\bar{y}$.

[11] Althofer 1994: On sparse approximations to randomized strategies and convex combinations *Linear algebra and its applecations* 1994; Lipton, Markakis, & Mehta: Playing Large Games using Simple Strategies. (extension from 2-player case to k-player case)

How big a support do you need (continued)

Suppose player 2 replaces y with empirical distribution $\bar{y}$ based on $N = O(\log(n/\epsilon^2))$ samples.

With high probability, each of player 1's pure strategies gets about the same payoff as before.

$$e_i^{\mathrm{T}} R\bar{y} = e_i^T Ry + O(\epsilon)$$

$\bar{y}$ has support $O(\log(n/\epsilon^2))$, so if we do the same thing with x we get the desired result.

We are using standard results about empirical values converging to true ones (use e.g. Hoeffding's inequality)

n random variables in $[0, 1]$; let S be their sum;

$$\Pr(|S - E[S]| \geq nt) \leq 2e^{2nt^2}$$

Support enumeration

Note that it follows that for any ϵ we can find ϵ-NE in time $O(n^{\log(n)})$.
(Pointed out in Lipton et al; another context where support enumeration "works" is on randomly-generated games[12])
Contrast this with **NP**-hard problems, where no sub-exponential algorithms are known. This is evidence that probably the problem of finding ϵ-NE is in **P**.

[12] Bárány, Vempala, & Vetta: Nash Equilibria in Random Games. *FOCS* '05

$k > 2$ players

Very little is known for $k > 2$.

- Constant support-size: we can achieve $\epsilon = 1 - \frac{1}{k}$ (equals $1/2$ for $k = 2$) but cannot do better.[13]
- this gets very weak as k increases!
- For 2 players, LP-based algorithms do better than $1/2$, but some new approach would be needed for $k > 2$.

[13] Hémon, Rougement & Santha: Approximate Nash Equilibria for Multi-player Games. *SAGT* '08, and independently, Briest, G, & Röglin: Approximate Equilibria in Games with Few Players. *arXiv* '08

2 players; improvements over $\epsilon = 1/2$

How to achieve $\epsilon \approx 0.382$: [14]

Recall (in DMP algorithm) player 1's initial strategy may be poor, but it doesn't help to pick a better **pure** strategy
Instead, pick a mixed one as follows

[14] Bosse, Byrka, & Markakis: New Algorithms for Approximate Nash Equilibria in Bimatrix Games. *WINE* '07; *TCS* 2010

2 players; improvements over $\epsilon = 1/2$

How to achieve $\epsilon \approx 0.382$: [14]

Recall (in DMP algorithm) player 1's initial strategy may be poor, but it doesn't help to pick a better **pure** strategy
Instead, pick a mixed one as follows
Original game is (R, C); solve zero-sum game $(R - C, C - R)$; let x_0 and y_0 be player 1 and 2's strategies in the solution

[14] Bosse, Byrka, & Markakis: New Algorithms for Approximate Nash Equilibria in Bimatrix Games. *WINE* '07; *TCS* 2010

2 players; improvements over $\epsilon = 1/2$

How to achieve $\epsilon \approx 0.382$: [14]

Recall (in DMP algorithm) player 1's initial strategy may be poor, but it doesn't help to pick a better **pure** strategy

Instead, pick a mixed one as follows

Original game is (R, C); solve zero-sum game $(R - C, C - R)$; let x_0 and y_0 be player 1 and 2's strategies in the solution

Let α be a parameter of the algorithm; if x_0 and y_0 are an α-NE use them, else continue...

[14] Bosse, Byrka, & Markakis: New Algorithms for Approximate Nash Equilibria in Bimatrix Games. *WINE* '07; *TCS* 2010

2 players; improvements over $\epsilon = 1/2$

Let j be player 2's best response to x_0; player 2 uses pure strategy j.

2 players; improvements over $\epsilon = 1/2$

Let j be player 2's best response to x_0; player 2 uses pure strategy j.
We can assume player 2's regret is at least player 1's.
Let k be player 1's pure best response to j; player 1 uses a mixture of x_0 and k.
Mixture coefficient of k is $(1-r)/(2-r)$ where r is player 1's regret in the solution to the zero-sum game.
Optimal choice of α is $(3-\sqrt{5})/2 = 0.382...$

2 players; improvements over $\epsilon = 1/2$

Proof Idea:

When player 2 changes his mind (from using y_0) he is to some extent helping player 1; y_0 arose from a game where player 2 tries to hurt player 1 as well as helping himself.

In the paper, they tweak the algorithm to reduce the ϵ-value down to 0.364.

Communication complexity

Uncoupled setting[15] of search for equilibrium: each player knows his own payoff matrix. Play proceeds in rounds (steps, periods, days). A player observes opponents' behaviour.

Communication complexity: question of how many steps are needed, where players don't need to follow a rational learning procedure.

n players, 2 action per player;[16] each player's payoff function has size 2^n: For exact NE, 2^n rounds are needed.

Obstacle is informational, not computational.

[15] Hart, S., Mas-Colell, A., 2003. Uncoupled dynamics do not lead to Nash equilibrium. Amer. Econ. Rev.

[16] Hart, S., Mansour, Y., 2010. How long to equilibrium? The communication complexity of uncoupled equilibrium procedures. *Games Econ. Behav.*

Communication complexity

2 players, n action per player: Search for pure NE, n^2 rounds are needed.[17] For exact mixed NE, $\Omega(n^2)$ rounds; polylog communication enough for ϵ-NE with $\epsilon \approx 0.438$[18]

Fun open problem: if 2 players cannot communicate, for what ϵ can ϵ-NE be found? (known to lie in $[0.51, 0.75]$)

[17] Conitzer & Sandholm, 2004: Communication complexity as a lower bound for learning in games. *21st ICML*

[18] G & Pastink (2014): On the communication complexity of approximate Nash equilibria. *GEB*

Query complexity

Algorithm gets black-box access to a game's payoff function: "payoff query" model[19] — algorithm can specify pure-strategy profile, get told resulting payoffs

Motivation:

- n-player games have exponential-size payoff functions; black-box access evades problem of exponential-size input data
- Amenable to lower bounds and upper bounds
- models "costly introspection" of players

[19] Introduced in: Fearnley, Gairing, G and Savani (2013): Learning Equilibria of Games via Payoff Queries. *14th ACM-EC*. Hart and N. Nisan (2013): The Query Complexity of Correlated Equilibria. *6th SAGT*; Babichenko and Barman (2013): Query complexity of correlated equilibrium. *ArXiv*.

Query complexity

Some results:

- For bimatrix games, QC is n^2 for find exact NE.
- ...to find ϵ-NE, $O(n)$ for $\epsilon \geq \frac{1}{2}$
- n-player games: exponential for *deterministic* algorithms to find anything useful; or for any algorithm to find *exact* equilibrium (Hart/Nisan)
- Query-efficient algorithms to find approx *correlated* equilibrium (Hart/Nisan; G/Roth)
- ...

Conclusion

Mainly focused on a particular sub-topic of AGT. *Algorithmic Game Theory* (2007) has 754 pages; and much has been done since!

Thanks for listening!